Deep Generative Models with Discrete Latent Codes for Text-to-Image Generation

Andrey Sukhobok

School of Science

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 02.09.2021

Supervisor and advisor

Prof. Alexander Ilin

Aalto University
School of Science

Aalto University
School of Science

Aalto University, P.O. BOX 11000, 00076 AALTO
www.aalto.fi
Abstract of the master's thesis

Author Andrey Sukhobok

Title Deep Generative Models with Discrete Latent Codes for Text-to-Image Generation

Degree programme Computer, Communication and Information Sciences

Major Machine Learning, Data Science and Artificial Intelligence **Code of major** SCI3044

Supervisor and advisor Prof. Alexander Ilin

Date 02.09.2021 **Number of pages** 49 **Language** English

Abstract

Text-to-image generation remains challenging and comprehensive task in the area of generative models. In this work, we considered a recently proposed approach, called DALL-E. This model is based on two popular neural network architectures - Discrete Variational AutoEncoder and Transformer.

The variability of possible Transformer configurations opens up an opportunity to explore the influence of different architectural choices on the model's performance. We concentrated on the way, how DALL-E processes the input sequence of text and image tokens. More specifically, we tried to check if there is any systematic advantage of using a separate text encoder instead of processing both data modalities (text and image) by the same autoregressive component (Transformer encoder). Additionally, we performed an analysis of different types of Discrete Variational AutoEncoders.

For the purpose of comparison between different Transformer components of the DALL-E approach, we created a specific dataset, that we called Multi-Descriptive MNIST. This dataset consists of the descriptions and corresponding images with sequences of digits with different characteristics, like color, size or location of the canvas. Also, we conducted some experiments on the CUB dataset, that consists of birds images with corresponding textual descriptions.

Finally, we developed a set of specific metrics, to compare the quality of the generated images. Since text-to-images task implies that text provides us the control over the generation process, we concentrated on the measurement of the consistency between the text and images, that are generated by this text. The idea behind the proposed metrics relies on the Contrastive Language–Image Pre-training (CLIP) model, that was recently introduced as a way to perform image classification based on the relevance between images and texts.

Based on the conducted experiments, we found a statistically significant advantage of using separate text encoder for the DALL-E approach over the original method on the specifically prepared artificial dataset. Also, the model with separate text encoder was trained on CUB dataset from scratch to generate images of birds consistent with the given text.

Keywords Deep Learning, Neural discrete representation, Text-to-image generation, Transformer, VQ-VAE, dVAE, DALL-E, CLIP

Contents

1 Introduction

During the past decade, Deep Learning has become the core technology in the area of artificial intelligence (AI). Neural networks have demonstrated their ability to solve a variety of complex tasks with a very high quality. Many of the services that people use everyday, like language translation apps or search engines, rely on this technology.

The area of deep generative modelling became extremely popular in research during the recent years. The general goal of such models is to synthesize the data, like images, audio or text, without strict guidance and big efforts from human. The ability to generate data of high quality would be an important achievement for both practical applications and research. On the one hand, creative professions, like designers, can use such models to speed up and enhance their work. On the other hand, model's ability of high quality data generation can indicate a high level of understanding of the data. So, it can help scientists to move closer to creating AI, that can understand and interact with our world.

One of the particularly popular tasks in the described field is text-to-image generation. Its formulation is straightforward: generate an image, that will correspond to a given textual description. One of the very widespread approaches to this task is the Generative Adversarial Networks (GANs). Such models consist of two parts, Generator and Discriminator, that compete in a two-player minmax game. Generator tries to synthesize an image, that would look like a real one and Discriminator tries to distinguish between the true images and the ones, provided by the Generator. Many works, like [13], [26], or [24] rely on this approach and demonstrate a high quality results on different datasets.

A new approach for text-to-image generation was recently introduced by OpenAI and showed very impressive results. The presented model is called DALL-E and rely on two common architectures – discrete variational autoencoder (dVAE) and Transformer.

Variational autoencoders is a widespread class of algorithms, that are used to compress the data in some latent representation. Different ways of such compression allow different ways of usage. For example, images can be modeled as samples from some prior distribution ([18]), like multivariate Gaussian, from which we can easily sample observations. However, such formulation doesn't give us much control over the hidden representation and the sampling process. VAE, used as a part of DALL-E model, uses another technique, that encodes an image with a set of discrete tokens. Works like [19] and [6] demonstrate not only the ability to reconstruct an image from the learned discrete tokens, but also to sample from this discrete latent space by learning a prior model, like PixelCNN, over the distribution of tokens sequence.

Another important part of the DALL-E architecture is a Transformer [7]. Originally, text related applications were the main focus of this model, where it demonstrated impressive results. Later, the application horizon expanded to computer vision ([10], [27], [20], [23]), especially, in pair with discrete image representations, since the input, formed by the discrete tokens, is a natural fit for the Transformer model.

One important achievement of the DALL-E model is its zero-shot generation quality. It was able to demonstrate competitive results on the datasets, that were not included in the training process, while compared to the models, that were specifically trained on this data.

A very general description of DALL-E can be written in a couple sentences. Firstly, the discrete VAE is trained to represent an image as a set of tokens from a vocabulary of fixed size. Secondly, a Transformer model is trained to convert given text tokens to the corresponding image tokens in autoregressive manner.

One particularly important architectural choice for the DALL-E model is that text and image tokens are processed as a single stream of data. Basically, text tokens are modeled as the beginning of a sequence of the image tokens. However, many successful Transformer-based sequence-to-sequence models ([21], [23]), including the original architecture [7] for machine language translation, use encoder-decoder design. The input sequence is usually encoded in some latent representation and, then, decoder processes the output sequence in autoregressive manner. Works, like [9], demonstrate the ways of speeding up Transformers by introducing new ways for calculating attention mechanism. So, the variability of possible Transformer configurations opens up an opportunity to explore the influence of such architectural choices on model's performance. Also, as it was mentioned above, the size of the training data is an important part of the results, provided by DALL-E model. However, researches and businesses quite frequently can't afford to collect such big datasets. Moreover, it is typical for the industry to be interested in one particular data domain. So, it might be reasonable to explore the performance of such model on a specific dataset with its own characteristics.

In this thesis, we explored several architectural questions regarding different parts of the approach offered by the DALL-E model. Firstly, we tried to check if there is any systematic advantage of using a separate text encoder instead of processing both data modalities (text and image) by the same autoregressive component. The performance of these models was measured on the domain specific dataset, which is closer to the real application of deep learning models, at least nowadays. This dataset was specifically created for this work and consists of images of digit sequences characterised by different parameters, like digit size, color and etc. Since the comparison of the generative models performance is not straightforward, we introduced a couple of metrics, that are based on the Contrastive Language–Image Pre-training (CLIP) model [22]. The main idea behind the introduced metrics is the estimation of the correspondence between the generated images and source texts. The details are provided in the main text of this work. Additionally to described comparison, we provide an overview and qualitative comparison between different types of discrete VAEs. Finally, we perform the training of the architecture with separate text encoder on the CUB dataset, that consists of birds images with the corresponding textual descriptions.

This thesis is organised in the following way: Section 2 provides an explanation of the relevant methods and architectures. Section 3 has a detailed description of the used datasets. Section 4 illustrates the difference between the models that we compare, describes the techniques that we use to train discrete autoencoders and

provides a detailed explanation of the metrics that we designed for models comparison. Experiments and results are presented in Section 5. It also contains the details of the training and architectural parameters of the considered models. Section 6 provides a summary and conclusions made from the experimental results.

The code is released at https://github.com/AndrewSukhobok95/discrete_latents_for_text_to_image_gen.

2 Methods and Background

2.1 Vector Quantised Variational AutoEncoder (VQ-VAE)

As it was mentioned above, AutoEncoder is a widespread class of algorithms in the area of Deep Learning, that compresses the data in some latent representation. At some point, Variational AutoEncoder (VAE) was introduced as a way to sample from this hidden representation. We will describe this sampling method more precisely a bit later, however, we will emphasise, that both VAE and traditional AutoEncoder work with continuous representation of the data.

The paper [19] introduces a type of such architecture, called VQ-VAE, that encodes input data with the discrete tokens. Another important difference from VAE is that learned model is used to simulate the prior distribution for the produced latent space instead of using a static prior distribution, like Gaussian or Uniform.

The subsections below describe the details of VQ-VAE model and provide an overview of different techniques for improving its training.

2.1.1 VQ-VAE Architecture

The model itself mainly consists of two parts: an encoder network, that approximates the posterior distribution $q(z|x)$ over the discrete random latent variables z given input data x, and decoder network, that approximates the distribution of x given the latent codes z. Additionally, we have a separate prior model $p(z)$, that is used to sample from the latent space.

Posterior and prior distributions are categorical, so the latent codes z are technically represented as an embedding vectors from the table of fixed size.

More formally, this table represents a latent embedding space $e \in R^{K \times D}$, where D denotes the dimensionality of the embedding vector and K defines the number of such vectors. The discrete latent variables z are computed by a look-up of the nearest neighbour from the output of the encoder network $z_e(x)$. The input of the decoder is a vector (or a set of vectors) extracted from the table e. Formally, it can be described as

$$z_q(x) = e_k, \quad \text{where} \quad k = \arg\min_j \|z_e(x) - e_j\|_2.$$

The illustration of the VQ-VAE architecture is presented in Figure 1. The spatial structure of the latent space produced by VQ-VAE is defined by the data domain and may be one-dimensional, two-dimensional or three-dimensional for speech, image or video, respectively.

Since, the look-up procedure doesn't have a way to estimate gradients, the authors just copy gradients from the decoder input $z_q(x)$ to the encoder output $z_e(x)$ during the backward pass. Equations 1, 2, 3, 4 define the overall loss for the training of

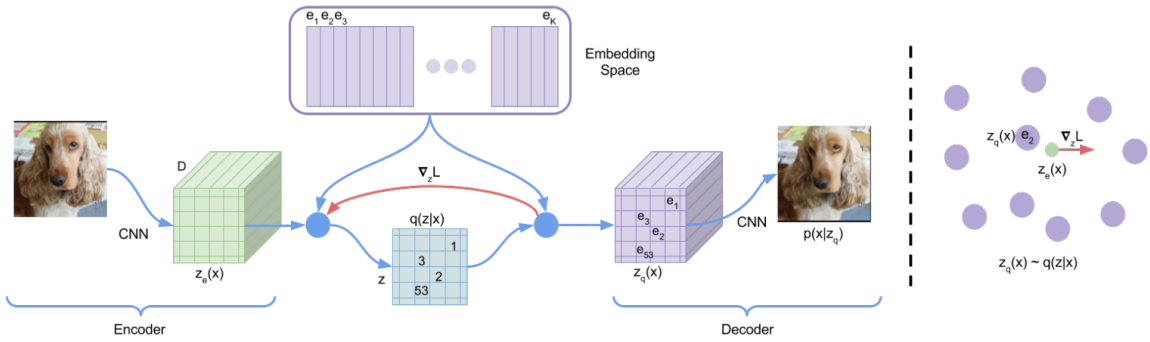


Figure 1: VQ-VAE Architecture. [19]

VQ-VAE.

$$L = L_1 + L_2 + \beta L_3 \tag{1}$$
$$L_1 = \log p(x|z_q(x)) \tag{2}$$
$$L_2 = \|sg[z_e(x)] - e\|_2^2 \tag{3}$$
$$L_3 = \|z_e(x) - sg[e]\|_2^2 \tag{4}$$

where $sg[.]$ represents the stop-gradient procedure and $\|.\|_2^2$ - an l_2 loss.

The first part L_1 is responsible for encoder and decoder weights updating through the mechanism of copying the gradients, explained above. However, it doesn't allow us to train the embeddings. That's why the vector quantisation technique is applied here by introduction of the second term L_2. This objective moves the embeddings towards the output of the encoder in the latent space. The last term L_3 is called commitment loss and tries to ensure that the volume of space formed by the output of the encoder doesn't grow arbitrarily.

To sum up, the decoder optimizes the first loss term L_1, the second term L_2 helps to learn the embedding space and encoder optimizes both L_1 and L_3.

The described method was applied to three data domains - images, audio and video. The model demonstrated a high level of its reconstruction abilities. Also, different autoregressive models were trained to approximate a prior categorical distribution $p(z)$ over the discrete representation formed by VQ-VAE. PixelCNN ([1], [2]) and WaveNet [3] architectures were used for image and audio data, respectively.

2.1.2 VQ-VAE Stable Training Procedure

Due to the non-trivial procedure of gradients estimation for the backward pass, the training of VQ-VAE is challenging. In practice, you need some tricks and additional steps to achieve good results. A comprehensive overview and evaluation of such procedures were proposed in [5]. Here, we will shortly formulate the main points of this work.

Firstly, authors make an assumption about the magnitude of the output vectors generated by encoder and codebook vectors used as discrete tokens. It can be

formulated as: *If the codewords are smaller in norm, the selection depends on the angular distance of the encoded representation and the codeword, and many codewords are used. However, if the codewords are larger in norm, all encoder outputs are likely to be assigned to the smallest codeword.*[1] In order to prevent the lack of utilization of the codebook, the authors propose to use batch normalization layer to enhance the magnitude ratio between the encoder outputs and codewords.

Secondly, it is noticed that codebook might not properly adapt to changes in the encoder outputs during training. It might lead to organization of codeword groups that are too close to each other, so only one of them is frequently used. In order to avoid such situation, it is suggested to periodically perform `k-means++` clustering procedure during training to reassign the described codeword groups with one vector.

Thirdly, this work shows that Exponential Moving Average (EMA) update rule for the codebook increase the stability of training. This approach was theoretically described in the original paper, however, wasn't tried. The idea behind it is following: for every codeword w_i there are n_i closest outputs $e_1, ..., e_{n_i}$ from the encoder. We update the codeword vector with the average code m_i normalized by the usage count N_i:

$$N_i = \gamma N_i + (1 - \gamma) n_i \tag{5}$$

$$m_i = \gamma m_i + \sum_{j}^{n_i} (1 - \gamma) e_j \tag{6}$$

$$w_i = \frac{m_i}{N_i} \tag{7}$$

where γ is a discount factor.

Also, it was noticed, that using a higher learning rate for the codebook can help to achieve better results.

2.2 Gumbel-Softmax

Another approach to represent the data as samples from some encoded discrete latent space rely on the Gumble-Softmax trick, proposed in [15]. This algorithm was used as a part of DALL-E architecture and, similar to its authors, we will refer to it as discrete variational autoencoder or just dVAE. Similarly to VQ-VAE, this model consists of encoder and decoder networks. However, instead of storing the table with discrete embeddings, this approach represents one token as just one-hot vector where index filled with one corresponds to the index of a chosen token.

In general, Gumble-Softmax trick addresses the problem of gradients computation for sampling from categorical distribution. This approach is heavily based on the idea of the Reparameterization trick. It is a core part of VAEs [18] and provides a way of dealing with the stochasticity of the sampling process from a continuous distribution. So, we will consider two mentioned tricks sequentially to get a clear explanations of all the details.

[1] Cited from [5].

The main idea behind the Reparameterization trick is decomposition of the stochastic sampling process as a linear combination of a deterministic and a stochastic elements. More precisely, instead of directly sampling an observation $z \sim \mathcal{N}(\mu, \sigma)$, we can say that this observation is defined by parameterized mean μ and variance σ on the one hand and a random noise ϵ on the other hand. Such dependence is formally described as

$$z = \mu_\theta + \sigma_\theta \cdot \epsilon \qquad \epsilon \sim \mathcal{N}(0, 1) \tag{8}$$

where μ_θ and σ_θ are mean and variance parameterized by θ.

The advantage of this approach is the possibility to compute gradients of mean and variance of the continuous distribution with respect to their parameters. Now, we will consider the application of this idea to categorical distribution.

The first piece of the discretization mechanism for dVAE is so called Gumbel-Max Trick. It again separates two parts of the sampling process: the deterministic and stochastic one. The deterministic part is presented as log-probabilities of the classes from the categorical distribution. The stochastic part is a noise generated from the Gumbel distribution.

$$z = \texttt{one\_hot}\left(argmax_i[g_i + \log \pi_i]\right) \tag{9}$$

where π_i is a class probability and g_i is a sample from the Gumble distribution.

Now, we have a way to draw samples during a forward pass computations. However, we still have an $argmax$ function, which is not differentiable. In order to overcome this issue, we replace the $argmax$ function with the $softmax$ function.

The temperature parameter τ controls a tie between one-hot encoded categorical distribution and continuous density. As $\tau \to 0$ the distribution is closer to discrete, while when $\tau \to \infty$ the distribution is closer to uniform. The calculated values for sample from a distribution with k classes are presented by an Equation 10.

$$z_i = \frac{\exp((\log \pi_i + g_i)/\tau)}{\sum_{j=1}^{k} \exp((\log \pi_j + g_j)/\tau)} \qquad \text{for } i = 1, ..., k. \tag{10}$$

where z_i is a probability of class in the generated sample (out of k classes), π_i is a class probability and g_i is a sample from the Gumble distribution.

2.3 Transformer and Attention Mechanism

The Transformer model was proposed in [7] as a sequence-to-sequence model. It consists of two parts: encoder and decoder. Encoder receives a sequence of observations $(x_1, ..., x_n)$ and provides a continuous hidden representation $(z_1, ..., z_n)$. Decoder generates an output sequence $(y_1, ..., y_n)$ in autoregressive manner producing one element per iteration. Decoder takes hidden representation $(z_1, ..., z_n)$ and previously generated elements as inputs during each iteration.

We will take a closer look at architecture of these components, however, firstly, the attention mechanism is going to be considered. It was originally introduced in [12] as an improvement for recurrent neural networks for sequence-to-sequence modelling. The Transformer model actively exploits this mechanism with several improvements.

2.3.1 Attention Mechanism

In general, the attention function receives three sets of vectors - queries, keys and values. The number of output vectors equals the number of queries. Output vectors are computed as a weighted sums of values. For each output vector the weights are calculated as a softmax combinations between a particular query and the keys. In practice, all calculations are conducted in the matrix form and can be formally expressed using Equation 11.

$$Attention(Q, K, V) = softmax\Big(\frac{QK^T}{\sqrt{d_k}}\Big)V \tag{11}$$

where Q and K are queries and keys matrices with a dimension d_k and V is a values matrix with a dimension d_v.

The described attention mechanism is called Scaled Dot-Product Attention by the authors of the paper and its difference from the one introduced earlier is the scaling factor $\sqrt{d_k}$. The idea behind scaling is, as it is described in [7], to prevent small gradients in the softmax function.

Another improvement over the standard attention, proposed for Transformer model, was so called Multi-Head Attention mechanism. The idea behind is calculating attention h times for linear projections of queries, keys and values, what can be done in parallel. The result is concatenated and linearly projected again to receive the output.

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^O \tag{12}$$

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \tag{13}$$

where $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$ and $W^O \in \mathbb{R}^{hd_v \times d_{model}}$.

The scheme of Scaled Dot-Product Attention is presented on Figure 2. It can be noticed, that there is additional block on the scheme called Mask. Masking mechanism allows to prevent attending certain positions in a sequence. In classic Transformer model, masking is used to preserve the auto-regressive structure of the model preventing leftward information flow inside the decoder.

Mask should be added to the input of the softmax function and consist of 0 and $-\infty$. After the softmax computation values, summed with $-\infty$, will be turned into 0. Others will be intact since they are summed with 0. The mathematical formulation is presented below.

$$Attention(Q, K, V) = softmax\Big(\frac{QK^T + Mask}{\sqrt{d_k}}\Big)V \tag{14}$$

2.3.2 Model Architecture

As it was described above, the model consists of encoder and decoder parts. We will start with describing the encoder part.

Encoder consists of N blocks. Each block receives an input x and passes it through two sequential sublayers - Multi-Head Attention and position-wise fully connected feed-forward network (FFN). Each sublayer is followed by a normalization layer [16] and

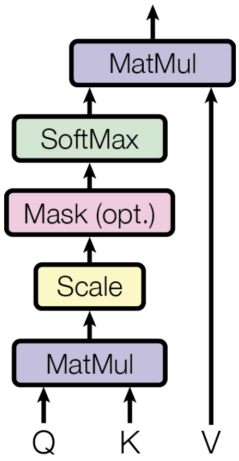


Figure 2: Scaled Dot-Product Attention. [7]

residual connection. Formally, it can be represented as $LayerNorm(x+Sublayer(x))$. The feed-forward sublayer consists of two linear layers with ReLU activation function. Formally presented as:

$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{15}$$

where W_1, W_2, b_1 and b_2 are trainable weights.

Decoder also consists of N blocks. Opposed to Encoder block, the Decoder's one consists of three sublayers. The first sublayer is Multi-Head Attention, that receives a subsequent mask in addition to the input key, value and query vectors. This mask hides future positions from the current ones, providing an auto-regressive structure of the decoder. Then there is one more Multi-Head Attention sublayer, which receives output from the previous layers as query and output from the encoder as key and value. Usually, the first attention is referred as self-attention and second one - as cross-attention. The last sublayer is the FFN similar to the one in Encoder block. In the end on the stack of decoder blocks there is a linear layer followed by a softmax non-linearity predicting a next element of the output sequence.

Additional detail of the model architecture is a so called Positional Encoding. Since this model doesn't imply any prior information about the structure of the input data, it uses a set of additional data to provide information about relative and absolute positions of the input elements. There are many position encoding approaches (fixed or learnable) [17]. The original paper uses sine and cosine encoding formally described by Equations 16 and 17.

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{model}}) \tag{16}$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{model}}) \tag{17}$$

where pos is the position and i is the dimension.

However, the original paper emphasises that learnable positional encoding provides almost the same results. Further papers like [9] or [10] showed positive impact of learnable embedding.

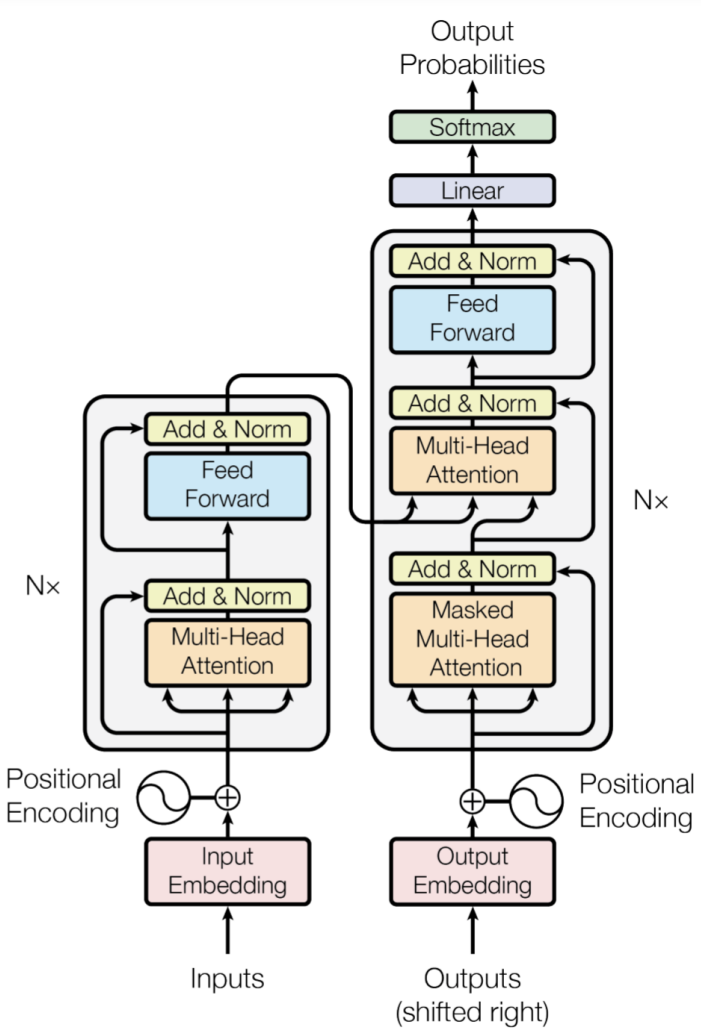


Figure 3: Transformer Architecture. [7]

The model showed impressive results for the Machine Translation Task on WMT 2014 dataset for English-to-German and English-to-French tasks. Additionally, good generalization abilities were shown on the English constituency parsing task.

2.4 Transformers for Computer Vision

After *Attention is All You Need* [7] paper, Transformers became extremely popular for different kinds of Natural Language Processing (NLP) tasks, like texts classification, machine translation, named entity recognition, syntax parsing and etc. So, this success raised a question, if the power of this architecture can be applied to an image domain.

The subsections below describe two recent papers, that demonstrate a high potential of Transformers for computer vision.

2.4.1 Vision Transformers

New work, called *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale* [10], published in 2020, introduced a way of using purely Transformer architecture without Convolutional Neural Networks (CNNs) for image classification task. The authors of the paper called the model Vision Transformer, or just ViT.

The general idea behind it is splitting the image into several regions and consider each region as an embedding, like it is common to do with words in NLP. The illustration of the model architecture is presented on the Figure 4.

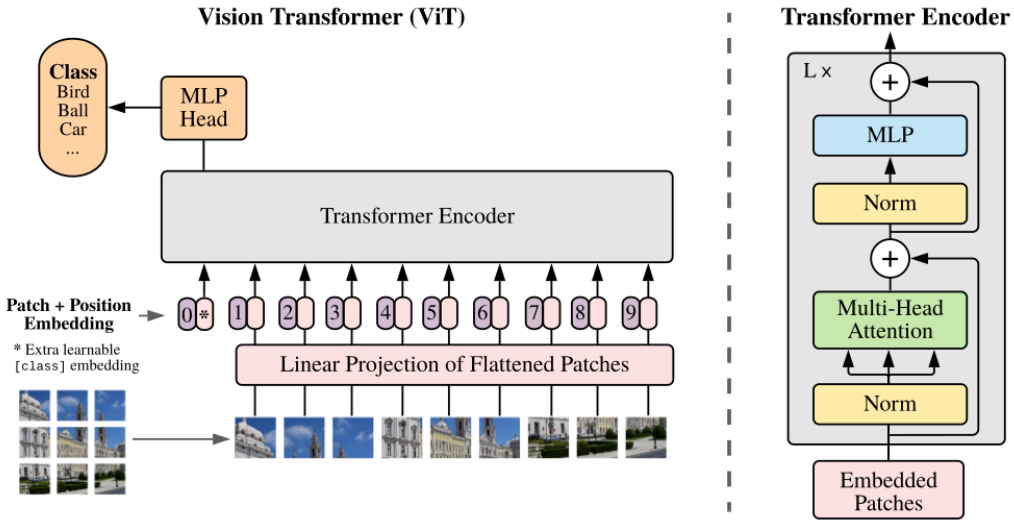


Figure 4: Vision Transformer Architecture. [10]

More formally, we can describe the way ViT works with its input in a following way: the image of shape (H, W, C) is firstly split into P by P patches (the shape of each patch is (P, P, C)). So, the total number of patches is $N = HW/P^2$, which corresponds to the length of the input sequence to the model. Since Transformer performs computations with the vectors of fixed size, the patches are firstly flattened and transformed to a vector of predefined dimension size using a trainable linear projection. Then the resulting sequence goes to the Transformer Encoder altogether with several modifications.

Additionally to the N tokens, received from the image, one trainable embedding, called `[class]` , is added in the beginning of the sequence. The purpose of this embedding is to serve as the hidden representation of the image in the output of the Transformer Encoder.

Also, learnable position embeddings, responsible for retaining the positional information, are added to the input tokens. The authors mention, that both one-dimensional (1D) and two-dimensional (2D) embeddings were experimented with. However, 1D type was finally used due to absence of significant difference in performance between two types.

Equations 19 and 20 show the architecture of the Transformer Encoder, that was used as the main part of the model. L stands for the number of blocks in the Encoder. Equation 18 shows the formal definition of the Encoder input. $\mathbf{E}$ stands for the linear transformation weight and $\mathbf{E}_{pos}$ stands for the position encoding weights. The output corresponding to the `[class]` embedding is later provided to the MLP classification head.

$$z_0 = [x_{class}, x_p^1\mathbf{E}, ..., x_p^N\mathbf{E}] + \mathbf{E}_{pos} \tag{18}$$

$$z'_l = MSA(LN(z_{l-1})) + z_{l-1} \quad l = 1, ..., L \tag{19}$$

$$z_l = MLP(LN(z'_l)) + z'_l \quad l = 1, ..., L \tag{20}$$

where $MSA(.)$ is a Multi-Head Attention layer, $LN(.)$ - Layer Norm layer and $MLP(.)$ consists of two linear layers with GELU activation function.

Similar to CNN architectures, the typical way of using the proposed Vision Transformer is with the fine-tuning step. It implies that the model is pre-trained on the large dataset and later fine-tuned on a smaller one. Paper [10] demonstrates a set of experiments with different model sizes and several famous datasets, including different versions of CIFAR and ImageNet. In terms of performance, ViT showed promising results, comparable to the state-of-the-art CNN architectures.

2.4.2 Generative Adversarial Networks with Transformers

Later, the idea of using pure Transformers for computer vision was extended in the paper *TransGAN: Two Transformers Can Make One Strong GAN* [27]. Its idea is straightforward - create a generative model for image synthesis using only Transformer architecture.

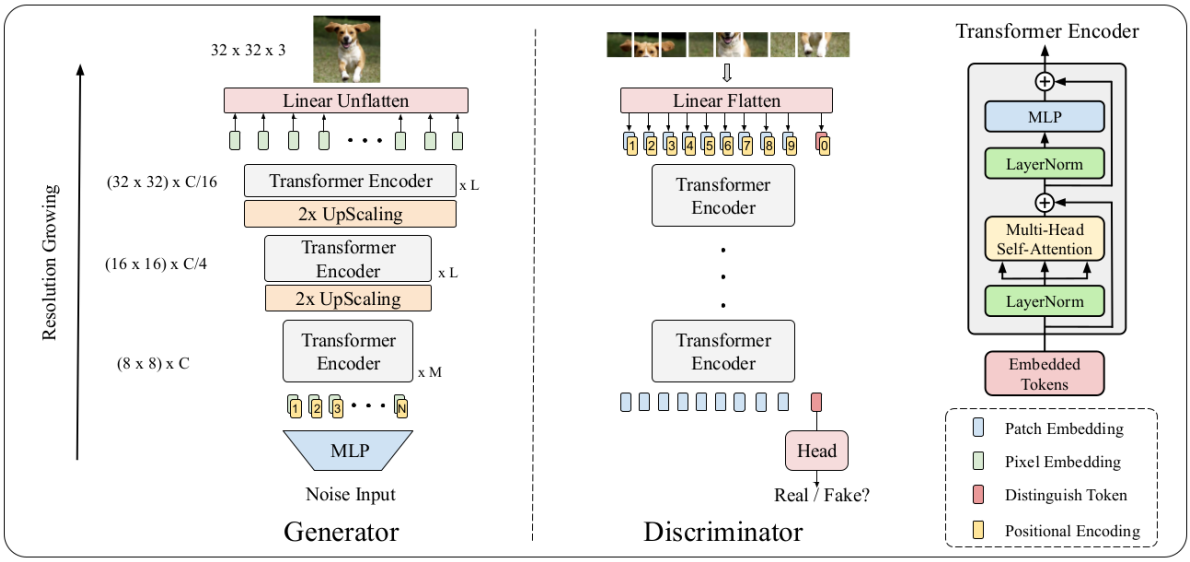


Figure 5: TransGAN Architecture. [27]

As a majority of GAN models, TransGAN consists of Generator and Discriminator. The general architecture for both of them is presented on Figure 5.

Since Transformer architecture is not specifically adjusted to grow an image similarly to CNNs, several additional steps were introduced. In the beginning of the Generator architecture, the random noise vector passes through a Multi-Layer Perceptron (MLP) and is being reshaped in the matrix, that represents the input tokens. Then tokens are added to the learnable position embeddings and the result goes to the Transformer Encoder. However, since Transformers are memory expensive, it will be quite heavy model, if all the tokens would be considered form the beginning. So, this work took inspiration from the growing structure of CNN GANs and introduced similar mechanism to this architecture. The generated image goes through two up scaling procedures, growing from 8×8 to 32×32 patches. The up scaling is performed through two steps. Firstly, tokens sequence is reshaped into two-dimensional feature map and, secondly, the resolution is increased through the pixelshuffle procedure [25]. Also, the number of channels decreases with the increase in the resolution. Finally, the desired output is linearly projected to three-dimensional space from which the RGB image is obtained. The Discriminator is entirely inherited from the Vision Transformer model. Similarly, the additional `[class]` token is used to distinguish between real and fake images.

However, the described choices for the model architecture are not enough to gain a competitive performance. So, the authors introduced several additional steps to achieve improvement. Firstly, experiments, conducted in the paper, suggest that data augmentation is more important for Transformer-based GAN, than for the CNN-based ones. Secondly, the technique called co-training was used, inspired by the importance of pre-training for Transformers in the NLP field. In general, co-training can represent any additional information, that we add to the loss, in order to help the model to find correspondence between this information and the goal, it tries to achieve. In case of the TransGAN, we add so called super-resolution loss, which is obtained as an MSE between generated high resolution image and low resolution image, returned from the middle stage of the Generator. This loss is added to the main one with an empirically determined coefficient. The last important detail is a locality-aware initialization for Self-Attention. Its mechanism is inspired by the inductive bias exploited by the CNN models. The mechanism of convolution allows such networks to look at local features at the beginning of the network and gradually increase its awareness of the global features by the increase of the receptive field. Similar thing is applied to the masks in the Self-Attention layers for TransGAN. Initially, masks are constructed in a way, that tokens can attend only to their neighbours on the 2D feature map. During training this replication of receptive field is increased until it achieves global attention on the later stages of the training.

To sum up, the described paper demonstrated very promising results in image generation on datasets CIFAR-10, STL-10 and CelebA using only Transformer architecture.

2.5 Transformers for Long Sequences Generation

Besides all the advantages and demonstrated success of Transformer architecture, there is still one weak point of this architecture - its computational complexity. That's

why one of the research directions is dedicated to exploration of the ways to make this model more efficient.

Paper [9] introduces a technique that allows to optimize the Transformer architecture in order to decrease the complexity of the model from quadratic to $O(n\sqrt{n})$.

The core idea of the paper is, so called, Factorized Self-Attention. It was inspired by the evaluation of the attention patterns performed on the CIFAR-10 dataset. The visualization showed that quite common patterns for images are to attend the previous pixels in the row or previous pixels in the column. So, these patterns were introduced to the transformer on the modelling level.

The input of the Factorized Self-Attention layer is a matrix of input embeddings X and the output is a weighted sum of input vectors transformed by a connectivity pattern $S = \{S_1, ..., S_n\}$. S defines which input positions will be attended by an output positions. These computations are formally described by the Equations 21, 22, 23 and 24, where $Attend(.)$ function provides the output of the layer. Matrices W_q, W_k, and W_v are responsible for transformation of x_i into query, key and value. d is a dimension of queries and keys.

$$Attend(X, S) = \Big(a(x_i, S_i)\Big)_{i \in \{1,...,n\}} \tag{21}$$

$$a(x_i, S_i) = softmax\Big(\frac{(W_q x_i) K_{S_i}^T}{\sqrt{d}}\Big) V_{S_i} \tag{22}$$

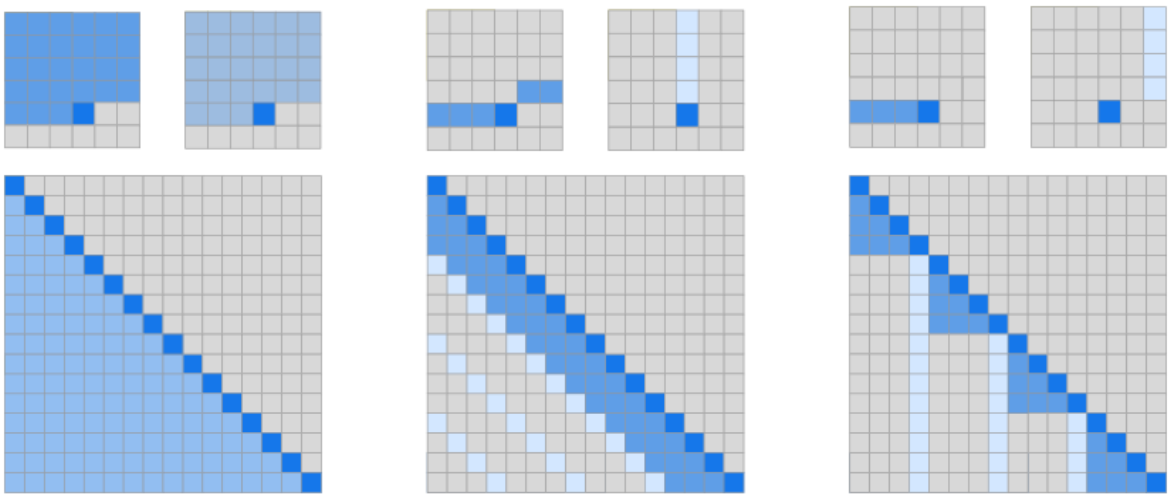
$$K_{S_i} = \Big(W_k x_j\Big)_{j \in S_i} \tag{23}$$

$$V_{S_i} = \Big(W_v x_j\Big)_{j \in S_i} \tag{24}$$

While full self-attention mechanism attends to all previous position and can formally be presented as $S_i = j : j \leq i$, the Factorized self-attention has p separate attention heads. The connectivity pattern of each head has to be built in a way, that all positions are connected across p steps of attention. This approach will allow to reduce computational costs and preserve the flow of information from arbitrary input positions to arbitrary output positions.

Paper [9] concentrates on the case with $p = 2$. In order to ensure attendance to all positions, the natural choice will be to have one head attending to previous l positions and another head attending to every lth position. Formally, such pattern can be presented as $A_i^{(1)} = \{t, t+1, ..., i\}$ for $t = max(0, i - l)$ and $A_i^{(2)} = \{j : (i - j) \ mod \ l = 0\}$. This method was called strided attention and showed to be convenient for the data with spatial structure.

However, for textual data, the paper introduced a better approach, which was called fixed attention pattern. The idea is that specific positions summarize the information from previous positions and pass this information further. Formal way of presenting this idea is $A_i^{(1)} = \{j : (\lfloor j/l \rfloor = \lfloor i/l \rfloor)\}$ with brackets indicating floor operation and $A_i^{(2)} = \{j : j \ mod \ l \in \{t, t+1, ..., l\}\}$, where $t = l - c$ and c is a hyperparameter. The strided and fixed patterns are presented in Figure 6 (b) and (c).



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Figure 6: Factorized attention schemes. [9]

The whole architecture is called Sparse Transformer with each residual block formally presented by Equations 25, 26 and 27.

$$a(H) = Dropout(Attention(Norm(H))) \tag{25}$$
$$b(H) = Dropout(FF(Norm(H + a(H)))) \tag{26}$$
$$resblock(H) = a(H) + b(H) \tag{27}$$

where $FF(X) = W_2 f(W_1 x + b_1) + b_2$ and $f(.)$ is a non-linear activation function.

However, there are several ways of defining the $Attention(.)$ function in this architecture. The simplest approach would be to use one of p heads per one residual block. In this case, the $Attention(.)$ function will be formulated as:

$$Attention(X) = W_p Attend(X, A^{(r \bmod p)}) \tag{28}$$

where r is the index defining the current residual block and p is a number of heads.

Another choice would be to have one head that allows attendance to all positions that both heads would attend to. The formal way is presented by the formula 29.

$$Attention(X) = W_p Attend(X, \bigcup_{m=1}^{p} A^{(m)}) \tag{29}$$

The last approach would be to use the multi-head attention mechanism with n_h heads and compute all of them in parallel. The results of each head are concatenated along the feature dimension axis.

$$Attention(X) = W_p \Big(Attend(X, A)^{(i)} \Big)_{i \in \{1,\dots,n_h\}} \tag{30}$$

Besides the Factorized Self-Attention layer, there are several additional improvements, that were introduced in the paper. Gradient check-pointing for reducing memory usage allowed authors to process the sequences of length 16384 tokens. For the fixed and strided attention patterns the computations were performed in blocks, since the introduced attention masks allow to localize the computations. This computation slicing technique increased the speed of training. Also, mixed-precision computations were used for training acceleration.

The Sparse Transformer architecture was tested on four datasets from different domains: CIFAR-10, EnWik8, ImageNet64x64 and the dataset of classical music from raw audio [11]. The equivalent or better results were shown for all of them in comparison with a standard Transformer architecture.

2.6 Zero-Shot Text-to-Image Generation (DALL-E)

All the techniques that we described so far were combined in the model called DALL-E, which we already mentioned several times throughout this work. Let us again state the main points behind this model architecture and describe the details of training process.

Paper [4] offered a new approach for text-to-image generation task.

The general idea of the method consists of two parts. Firstly, the algorithm learns how to represent the image by a predefined number of tokens from a vocabulary of fixed size using a discrete variational autoencoder (dVAE). Then the Transformer is used to generate this latent representation from text in autoregressive manner.

Additional advantage of this paper is a big dataset size (250 million image-text pairs collected from the internet), which allowed to achieve high quality image generation zero-shot without additional training labels.

The overall training procedure can be considered as evidence lower bound (ELB) maximization of the joint likelihood of the distribution over images x, their tokens representation z and corresponding text captions y. This distribution can be decomposed as $p_{\theta,\psi}(x, y, z) = p_\theta(x \mid y, z)p_\psi(y, z)$, from which the lower bound can be derived as:

$$\ln p_{\theta,\psi}(x, y) \geq \mathbb{E}\Big(\ln p_\theta(x \mid y, z) - \beta D_{KL}(q_\phi(y, z \mid x), p_\psi(y, z))\Big). \tag{31}$$

To put it in simple terms, we can say, that:

- q_ϕ represents the encoder of dVAE, that convert the given RGB image to a tensor of discrete codes.
- p_θ represents the decoder of dVAE, that receives the tensor of discrete codes and generate an RGB image.
- p_ψ represents the Transformer model, that receives text tokens and generate image tokens.

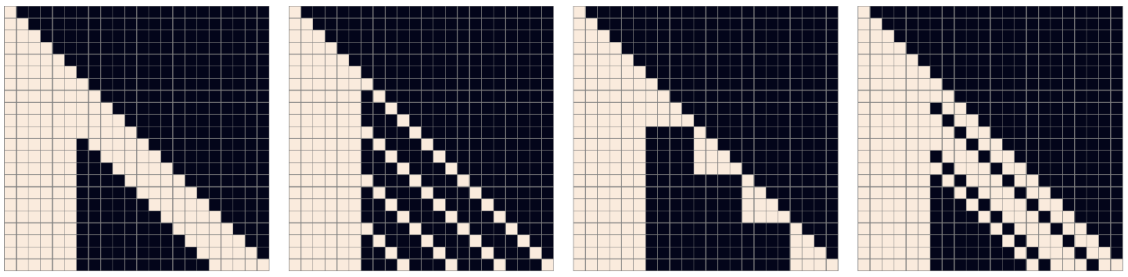
There are a couple of data processing details, that are important to mention. Firstly, the input text was BPE-encoded in order to fit the size of the caption tokens

to 256 positions. Secondly, dVAE was trained to compress 256×256 image into 32×32 grid of tokens. The overall number of possible tokens in the vocabulary is 8192.

As it was mentioned above, the algorithm has two stages that are trained separately.

Firstly, the dVAE is trained on images by ELB maximization with respect to ϕ and θ. The Gumble-Softmax was used as a quantisation algorithm instead of VQ-VAE. The temperature parameter τ was gradually annealed during training to improve divergence. Also, several additional tricks were used to enhance the stability of training. The β coefficient for the KL-divergence was gradually increased during training. Also the log-laplace distribution was used for evaluation of p_θ.

The second stage is training the prior distribution over the image tokens and text captions by ELB maximization with respect to ψ. The model architecture is a Sparse Transformer similar to the one introduced in [9]. Three type of attention masks were used in the model: row, column and convolutional. The example of such masks for a model working with text captions with maximum length of 6 tokens and image latent representation of 4×4 grid is presented on Figure 7. Masks (b) and (c) from Figure 7 are both column attention masks, however, the (c) one was used due to better GPU utilization. And mask (d) corresponds to 3×3 kernel, however, in real model the kernel of size 11×11 was used.



(a) Row attention mask. (b) Column attention mask. (c) Column attention mask with transposed image states. (d) Convolutional attention mask.

Figure 7: Illustration of the three types of attention masks. [4]

As it was mentioned above, the text caption is converted to 256 BPE-encoded tokens, that are considered to be the beginning of the predicted sequence. So, the first 256 tokens in the attention mask have full attention pattern. Since each caption consists of different number of tokens, it always padded to 256 length with a special learnable padding embedding. Also, both text and image parts of the sequence are summed with a position embeddings. While the text uses the one-dimensional positional information, the image has two-dimensional encoding of the positional information. The illustration of the described scheme for a hypothetical caption of length 6 tokens is presented on Figure 8.

In addition to all the described features and tricks this work also uses Mixed-Precision training and distributed optimization with a sophisticated manner of gradients calculation on the cluster of GPUs. Finally, drawn samples are reranked according to the pretrained contrastive model called CLIP [22]. Authors called this

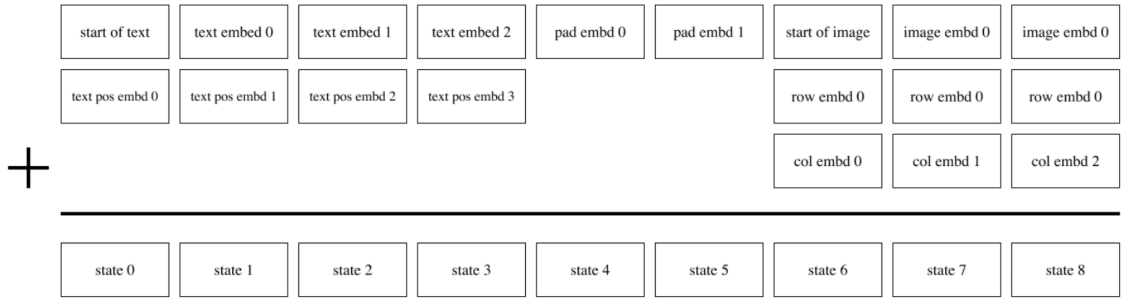


Figure 8: Illustration of tokens positions for the input of the Sparse Transformer model. [4]

reranking procedure as *a kind of language-guided search*[2]. The examples of the top image selected from the N best ones demonstrate that such guidance might be quite important for the quality of result.

The general outcome of the paper is a great demonstration of a potential for the usage of an approach, that combines discrete representations with Transformer model.

2.7 Learning Transferable Visual Models From Natural Language Supervision (CLIP)

This work [22] was published simultaneously with *DALL-E* paper [4] and exploited by it for reranking procedure, as it is mentioned above. In its core, it proposes the method for zero-shot image classification based on comparison between given text captions and images.

In general, the idea of using natural language supervision for image representation learning has a powerful motivation - large amount of publicly available text-image pairs in the internet. Previous works in this direction tried to solve the task of predicting the text that describes a given image. However, this task is particularly difficult due to the variety of possible descriptions for one image. Also, recent works showed that contrastive objectives allow to learn a better representations.

The Contrastive Language-Image Pre-training (CLIP) model solves an easier task: given a set of image-text pairs of size N, CLIP tries to define the correct pairs across $N \times N$ possible ones. The schematic representation of the whole approach is presented on Figure 9.

The architecture for the described approach consists of three parts: image encoder, text encoder and comparison their outputs. Image and text encoders are jointly trained to maximize the cosine similarity between real text-image pairs and minimizing between other $N^2 - N$ incorrect pairs. The outputs of the encoders are linearly transformed into the multi-modal embedding space.

The objective is a symmetric cross-entropy loss over the similarity scores. It can

[2]Cited from [4]

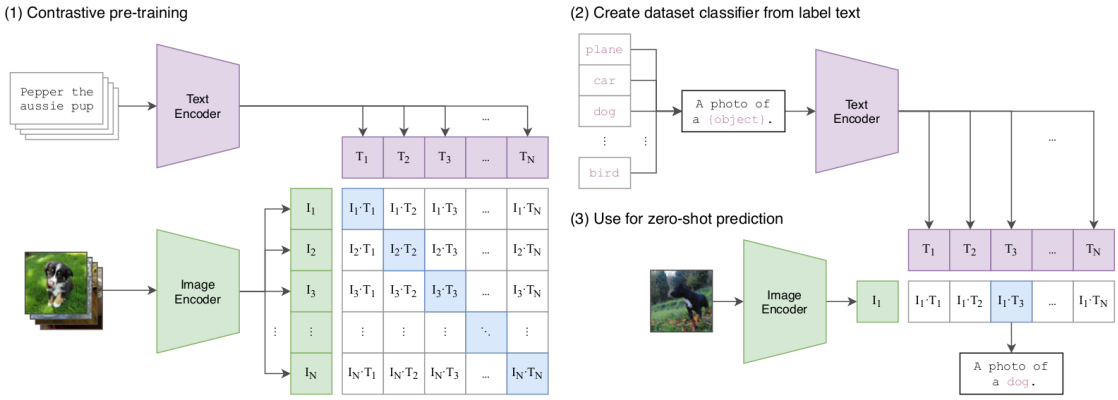


Figure 9: Summary of CLIP approach. [22]

be formally described by Equations 32, 33, 34 and 35.

$$Loss = \frac{L_{image} + L_{text}}{2} \tag{32}$$

$$L_{image} = CrossEntropy(logits, labels, axis = 0) \tag{33}$$

$$L_{text} = CrossEntropy(logits, labels, axis = 1) \tag{34}$$

$$logits = X_{image} X_{text}^{T} \cdot e^{\tau} \tag{35}$$

where X_{image} and X_{text} are normalized encoders outputs, *labels* are correct labels, that can be expressed in `numpy` notations as `np.arange(n)` for n pairs and *CrossEntropy* stand for the cross-entropy loss. `axis` parameter regulates whether the loss calculated across columns or rows. τ is a trained temperature parameter optimized during training as a log-parameterized multiplicative scalar.

12-layer 512-wide Transformer with 8 attention heads was used for text encoding. The text was lower-cased and transformed using byte pair encoding (BPE) with vocabulary size 49152 tokens.

As for images, two architectures were experimented with - ResNet [14] and ViT [10]. Both, image and text encoders, were trained from scratch without initialization with pretrained weights.

Since the dataset size is particularly important, the authors constructed a new dataset of 400 million image-text pairs collected from the public sources in the Internet. The dataset is called WebImageText or just WIT. In order to apply the trained model on the new dataset, the classes in the dataset can be described as a simple caption "*A photo of a {label}.*", where *label* represents the class of interest.

Zero-shot classification ability of the proposed model was tested on 27 different datasets. CLIP outperformed Linear Probe on ResNet50 on 16 of them, including ImageNet.

3 Data

3.1 Multi-Descriptive MNIST (MD-MNIST)

This dataset was artificially created from the classic MNIST dataset for the purpose of this work. In general, this dataset consists of pairs of images and their descriptions. So, we called it Multi-Descriptive MNIST, or MD-MNIST for short.

Each 128×128 image has a black background and three digits of different characteristics on it. These characteristics are provided in the description corresponding to the image and include the digit itself, its size, color and location on the canvas.

Similar to MNIST there are 10 possible digits - from 0 to 9. There are three available sizes corresponding to each digit - 20, 30, 40. These numbers represent the number of pixels to which the initial square image from MNIST will be resized to. There are four possible colors for each digit - *w, r, b, g* corresponding to white, blue, red and green. The location for each digit is chosen inside "its column", so that the first digit is in the left part of the canvas, second one in the center and the last one is in the right part. So, this characteristic influences only the vertical position of the digit. There are three possible positions - *up, middle and down.*

During the creation of the dataset each digit and its characteristics are chosen randomly. The created dataset has 100000 observations. Four examples from the generated dataset are presented on the on Figure 10. Each example consists of the image (on the right) and its description (on the left). The sequence of parameters in the description is always the same and corresponds to the following order - value, size, color, position.

Figure 10: Example from MD-MNIST dataset.

3.2 Caltech-UCSD Birds-200-2011 (CUB-200-2011)

CUB-200-2011 is created from the photos of the birds of different species. Each photo is accompanied by the class of the bird, several descriptive characteristics and bounding box of its location. In general, 200 categories of birds are presented in the dataset. The number of photos is 11788.

In this work we used the same captions[3], as the ones used in paper *AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks* [26]. Each image corresponds to a set of five short similar descriptions created from the image attributes, mentioned above.

Before using the text and images in the model, some preprocessing steps were performed.

As for the text, for every train sample, one of the five descriptions was chosen randomly to form a pair of image and text. Then, the text was tokenized using the following regular expression: \b \w+ \b.

Regarding images, one of the necessary steps is to reshape every image to a fixed size. In our experiments, every image was reshaped to $N \times N$ size, where N took values of 64 and 128. Also, different preprocessing procedures were chosen for the training of the autoencoder and for the training of the generative algorithm. These procedures are described below with reasoning behind their differences.

Type 1 preprocessing steps for the autoencoder training:

- Resize of the image to $(N + 10) \times (N + 10)$ pixels
- Random rotation of the image by 2 degrees
- Random crop of size $N \times N$ pixels
- Random horizontal flip of the image

Steps like rotation and flipping enhance the variability inside the dataset. Also, first three sequential steps (resize to $N + 10$ pixels, rotation and cropping) ensure that rotation will not lead to black triangles on the borders of the processed image. The sample of 16 images from the dataset with applied preprocessing steps of type 1 are presented on Figure 11.

Type 2 preprocessing steps for the generative model training:

- Square padding (the image is symmetrically padded on the side of the longer border)
- Random rotation of the image by 2 degrees
- Resize of the image to $(N + 20) \times (N + 20)$ pixels
- Center crop of size $N \times N$ pixels
- Random horizontal flip of the image

[3]The textual descriptions of the images were downloaded from the official github repository of the paper [26]: https://github.com/taoxugit/AttnGAN

Figure 11: Sample of images from the CUB-200-2011 dataset with applied preprocessing steps of type 1.

In case of the generative model, these steps are slightly different. The idea behind the second type of preprocessing steps is to ensure that the position of the bird is in the center of the image or close to it. Such approach makes sense for the generation process, since it is reasonable to generate the object of interest in the center of the created image. However, for the training of the autoencoder with discrete latents this property is not very important. Even if a half of the object will be cut from the picture, the main purpose of the autoencoder is to reconstruct the given image. So, the corruption of the first type, described above, may even help training to ensure variability of the training data. The sample of 16 images from the dataset with applied preprocessing steps of type 2 are presented on Figure 12.

Figure 12: Sample of images from the CUB-200-2011 dataset with applied preprocessing steps of type 2.

Figures 11 and 12 demonstrate the difference between two preprocessing procedures. Even on the sample of 16 images, type 1 has 6 images, where bird is partially cut from the picture.

4 Approach

This section provides a detailed explanation of what models were compared in this thesis and which methods were used to provide quantitative estimation of the results. Firstly, we give a short summary of the types of autoencoders we consider in this work. Secondly, we describe the differences between the Transformers architectures, that were tested on the dataset, that we specifically designed for this work. We call it Multi-Descriptive MNIST (MD-MNIST) and its detailed explanation will be provided in the next section. Finally, we discuss the possible approaches to measure the quality of the generated images and provide an explanation of the CLIP-based metrics, that we used.

4.1 Discrete AutoEncoder Design

In Section 2 we described two choices for the discrete AutoEncoder architectures: VQ-VAE and dVAE.

The first one uses a look-up operation to access the discrete tokens from the vocabulary. The size of the vocabulary and tokens dimension are parameters to choose. The encoder's and decoder's weights are updated through a copying mechanism, that copies gradients from decoder input to encoder output during the backward computation. Additionally, there are several hacks, that can be used to improve training. The list of the ones that we tried, is provided in the experiments.

dVAE uses the Gumble-Softmax trick to sample from a discrete distribution. Each token is a one-hot vector, so, this approach doesn't store the table with learned embeddings. Therefore, the bigger vocabulary size you use, the bigger channel dimension of the latent image representation you receive.

4.2 Transformers for Image Generation: Design Choices

As it was stated above, one of the goals of this work is to figure out whether there is a difference in performance between two architectures of Transformer. We consider the schematic representation on Figure 13 to be the best explanation of the differences between two models. The left scheme represents the DALL-E style architecture. Text tokens are marked by green color and image tokens, received from dVAE, are marked by blue color. The dark blue token represents a start-of-sequence, from which we start the autoregressive process during inference. The right scheme architecture represents the alternative architecture with separate text encoder.

4.3 Metrics and Architectures Comparison Methodology

For evaluation purposes we use three metrics, that are described in this section.

4.3.1 Evaluation by Loss

The first straight-forward way to compare two autoregressive generative models is to compare their performance in terms of their main training goal - predicting the next

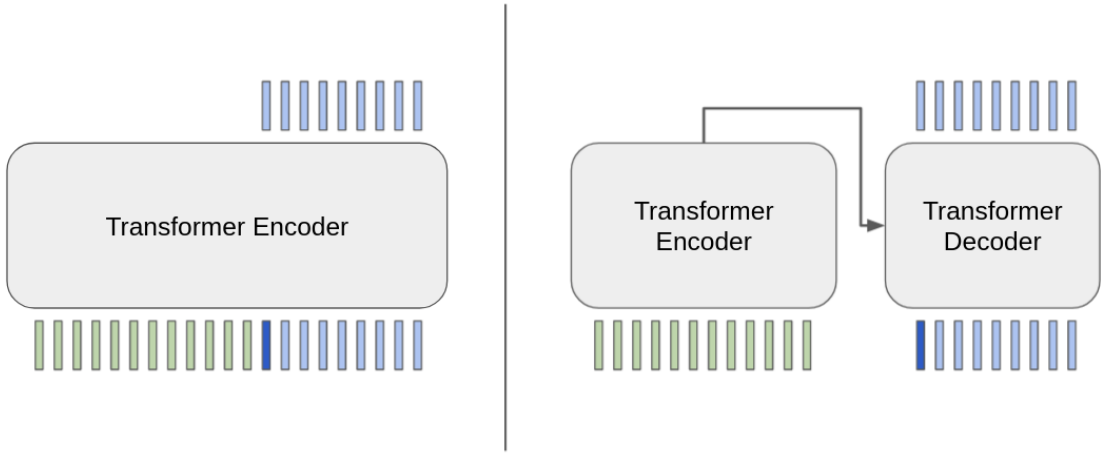


Figure 13: Difference between proposed architectures. The toy example on the figure assumes that caption consists of 12 tokens (green color) and image is encoded with 9 tokens (blue tokens). The start token for image has dark blue color.

tokens given all previous ones.

For this purpose, we propose the following setup: for a given number of observations we run the model the same way we do during training. Given the image and its textual description, the trained model predicts the same image in autoregressive manner. The outputs are compared with the correct ones in terms of the cross-entropy loss function. Since we assume, that we use significant amount of observations for such comparison, some aggregation over computed loss values is needed. The exact aggregation methods, performed in the work, are discussed in the experiments section.

However, such approach is biased and not very reliable. Firstly, cross-entropy loss is directly optimized during training. Secondly, it is not really interpretable metric, so it is difficult to make conclusions from it. And, most importantly, it doesn't really tell us how well the generated image matches the text, from which it was generated.

So, we need an interpretable and independent of our models way to estimate our results. That's the goal that next two approaches try to accomplish.

4.3.2 Evaluation by CLIP-based Images Comparison

One of the simplest ways to determine a better model is to show the pairs of texts and generated images to a group of people and ask them to choose which one is better. However, such way is expensive and non-realistic for the scope of this work. So, we need some kind of independent evaluator to choose between generated images.

We will use the CLIP model to simulate human evaluation, that is based on the choice of better generated image. As it is described above, this model takes images and texts and finds related pairs. We will use the following scheme: two generative models produce two images based on the same text. Then these two images and text are given to a CLIP model, which defines what image corresponds to the text better. The number of preferences of one model over another in a sufficiently large number of

experiments can be considered as quantitative way to compare two (or more) models. We will further call this metric CICS, as short for CLIP Image Comparison Score.

The schematic representation of CICS metric calculation is presented on Figure 14.

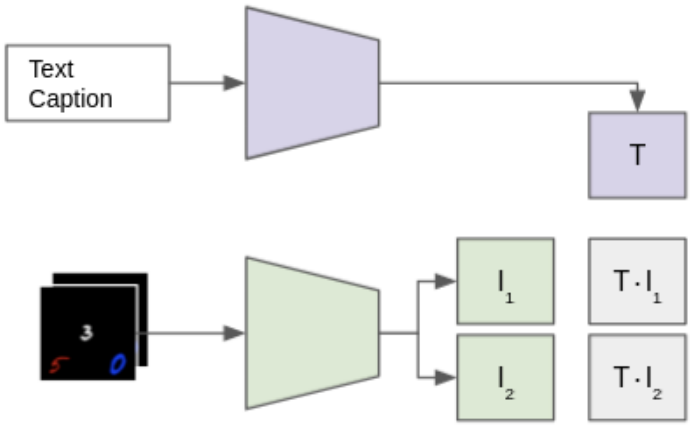


Figure 14: Schematic representation for CICS.

4.3.3 Evaluation by CLIP-based Image Ranking with Text

The CICS metric gives us a way to estimate which one of the generated images corresponds to a text better. However, this metric is not very descriptive. It doesn't really account for a situation, when both images are bad, but one of them is a little bit better than another. So, we need a way to estimate how good the image itself matches a given text. That's why we introduce one more metric.

The idea behind it is following: firstly, we generate an image from a given text. Then, we create several more texts from the original one using a special technique. In every next modification we replace more tokens for a different ones than in previous modifications. Intuitively, it means that every next modified text is further from the original one in terms of image correspondence. Finally, we can run the generated image and all the texts (including the correct one) through the CLIP model, which will tell if any of the modified text versions is closer to the image than the true one. If we perform such operation on sufficiently large number of examples, we can look at the distribution of mistakes. Such approach can be used to estimate the models themselves and compare them between each other.

The result of the described procedure will be called CTRS, as short for CLIP Text Ranking Score.

In case of the MD-MNIST dataset, which will be entirely described in the next section, each image has 12 characteristics, which can be provided to the model in a form of tokens similarly to text. So, for one image we can create 12 modifications of its original description. First modification will be different by one random token,

which will be replaced by the other random token different from the original one. The second modification will be different by two tokens and so on.

It is important to mention, that this approach is easy to implement for an artificial dataset, like MD-MNIST. However, in case of a real dataset with textual descriptions, like CUB, it would be a bit more difficult task. We will need a set of rules, that will define how text is modified. For example, the adjectives have to be replaced with adjectives and etc. Alternatively, the text can make no sense after the random replacements and it will be too easy for the model to distinguish the correct text from the modified versions.

The schematic representation of CTRS metric calculation is presented on Figure 15.

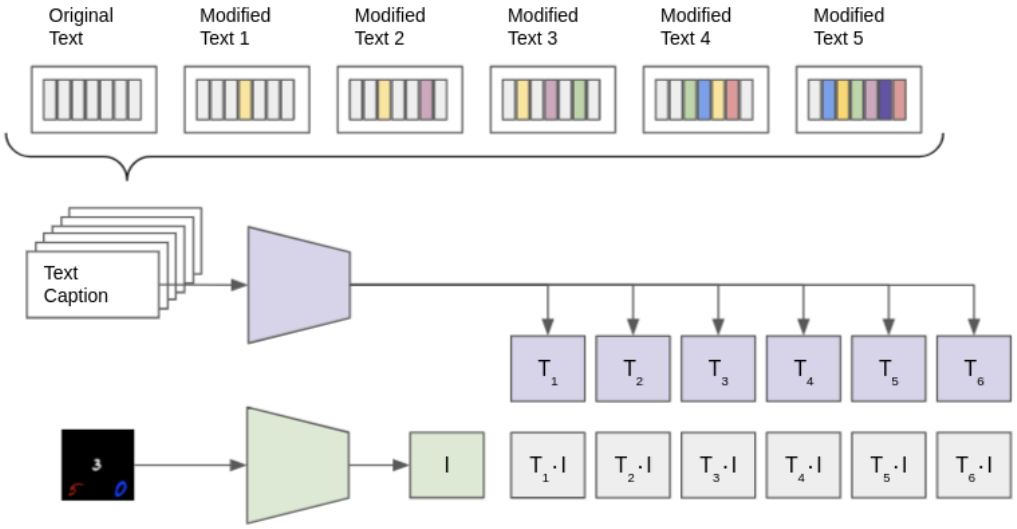


Figure 15: Schematic representation for CTRS.

5 Experiments and Results

In this section, we describe and discuss the performed experiments and the received results. The text is organised in four subsections. The first one introduces the notations and terms, which will allow us to easily describe the architectures of our models in subsequent subsections. The second one provide the comparison between different autoencoders. The third subsection shows the qualitative results of image generation made by Transformer architectures described above. Results are provided for both datasets - MD-MNIST and CUB. The last subsection performs the quantitative analysis of generated MD-MNIST images using the metrics introduced earlier.

5.1 Models Architectures

We constructed all models in our experiments from the simpler blocks, which can be easily described. In this section we will provide the architecture description for all these blocks using PyTorch[4] notations, that are listed below.

- The linear feed-forward layer will be defined as $Linear(H_{in}, H_{out})$ with H_{in} in features and H_{out} out features
- The convolutional layer will be defined as $Conv(C_{in}, C_{out}, K, S, P)$ with C_{in} in channels, C_{out} out channels, kernel size K, stride S and padding P
- The transpose convolutional layer will be defined as $TConv(C_{in}, C_{out}, K, S, P)$ with the similar notations as $Conv(C_{in}, C_{out}, K, S, P)$
- The dropout layer will be defined as $DropOut(P)$ with probability P
- The activation layers will be defined as $ReLU()$, $GELU()$ and $Softmax()$

5.1.1 Autoencoders

The basic blocks are Residual, UpSampleX2 and DownSampleX2. DownSampleX2 block uses convolution layers to increase the height and width of the image by a factor of 2. UpSampleX2 provides a reverse operation by using transposed convolution layers. The structure of these blocks is presented in Table 1.

Additionally, we used a ChangeChannels block, to change a number of channels in the beginning of the encoder or end of the decoder. The structure of this block is presented in Table 2.

The channels are written as variables in the descriptions above, because they are defined by the encoder and decoder architecture. We applied the following rule: we set the hidden dimension of the encoder or decoder to H. In the beginning of the encoder, we start with the *ChangeChannels* block to convert number of channels to $H/2^{Num.DownSamples}$, where $Num.DownSamples$ is a number of downsamples of the

[4]Python package for deep learning https://pytorch.org/

Residual Block	UpSampleX2 Block	DownSampleX2 Block
ReLU()	TConv(C_{in}, C_{out}, 4, 2, 1)	Conv(C_{in}, C_{out}, 4, 2, 1)
Conv(C_{in}, C_{out}, 3, 1, 1)	BatchNorm(C_{out})	BatchNorm(C_{out})
BatchNorm(C_{out})	ReLU()	ReLU()
ReLU()	TConv(C_{out}, C_{out}, 3, 1, 1)	Conv(C_{out}, C_{out}, 3, 1, 1)
Conv(C_{out}, C_{out}, 1, 1, 0)	ReLU()	ReLU()

Table 1: The architecture of basic blocks: Residual, UpSampleX2 and DownSampleX2.

ChangeChannels
Conv(C_{in}, C_{out}, 3, 1, 1)
BatchNorm(C_{out})
ReLU()
Conv(C_{out}, C_{out}, 1, 1, 0)

Table 2: The architecture of basic block ChangeChannels.

image height and width by a factor of two. Then throughout the encoder the out channel is multiplied by 2 every downsample step until it reaches H. The decoder uses similar rule applied backwards.

Encoder's and decoder's architectures for the dVAE model are shown in Table 3. N_{RD} and N_{RB} are hyperparameters determined for each particular experiment. N_{RD} defines the number of residual blocks after every downsample and upsample block. N_{RB} defines the number of residual blocks before the bottleneck.

Encoder	Decoder
ChangeChannels	ChangeChannels
DownSampleX2	UpSampleX2
$N_{RD}\times$ Residual	$N_{RD}\times$ Residual
DownSampleX2	UpSampleX2
$N_{RD}\times$ Residual	$N_{RD}\times$ Residual
$N_{RB}\times$ Residual	$N_{RB}\times$ Residual
ChangeChannels	ChangeChannels

Table 3: Encoder's and decoder's scheme for dVAE model used in the work.

5.1.2 Transformers

The basic blocks of Transformer are Encoder and Decoder blocks. In our experiments we used the same architectures as the ones used in ViT [10].

The basic layers of Transformer are $LayerNorm$, $MultiHeadAttention$ and $DropOut$. Encoder architecture is described by formulas 19 and 20. The decoder

architecture uses the same approach, however, its architecture is closer to original Transformer decoder block. There are two attention sublayers. First one uses only the input tokens as keys, values and queries to perform self-attention. Second sublayer takes output from the previous layers as query and output from the text encoder as key and value.

The structure of the Transformer's MLP block is presented in Table 4.

MLP for Transformer
$Linear(H_{in}, H_{hidden})$
$DropOut(P)$
$ReLU()$
$Linear(H_{hidden}, H_{out})$

Table 4: The architecture of MLP block in Transformer architecture.

5.2 Discrete AutoEncoder Experiments

The crucial part of the current work is training the autoencoder with discrete latents. As it is described above, there are two popular architectural choices - VQ-VAE and dVAE. The last option was used in [4] as a part of, so called, DALL-E model. However, both choices were explored.

Regarding VQ-VAE model, we used several techniques, described in [5], in order to improve the results. Firstly, encoder and decoder have Batch Normalization layers in their structure. The vocabulary codewords were updated using exponential moving average technique. Also, separate learning rate for vocabulary embeddings was particularly important for codebook utilization.

As for the dVAE model, two experiments are described in this section. The difference between them is an additional term in the loss function. The common part is binary cross entropy between the scaled pixels of true image and its reconstruction. Additional term is the KL-divergence for the distribution of codewords used in the current batch. The target distribution for the second term is uniform. The idea behind it is simple - enhance the vocabulary usage.

The training parameters for the described models are presented in Table 5. The Adam optimizer from the PyTorch package was used for all the described models.

In addition to the four mentioned models, we added the parameters for one more training procedure of dVAE for images 64×64 (instead of 128×128). As it will be shown later, the described algorithms require significant amount of training time. That's why we used an image of lower resolution to get better results faster for both reconstruction and generation, which will be discussed in the next section.

There are some additional notes, we need to make about the training parameters described in the Table 5. Firstly, in order to make two architectures comparable, the vocabulary size parameter is responsible for both number of tokens and their dimension in case of VQ-VAE model. Secondly, both temperature parameter and KL-divergence loss coefficient are both linearly changed through the stated number

	dVAE	dVAE growing KL	VQ-VAE	VQ-VAE sep. vocab.	dVAE 64×64
Num. epochs	150	150	150	150	500
Batch size	16	16	16	16	64
Learning rate (LR)	0.001	0.001	0.001	0.001	0.001
LR vocabulary	-	-	0.001	0.01	-
LR γ	0.1	0.1	0.1	0.1	0.1
LR milestones	5,100	5,100	5,100	5,100	20,120
Vocab. size	512	512	512	512	512
τ* start value	5	5	-	-	8
τ* end value	0.01	0.01	-	-	0.01
τ* iterations	80000	80000	-	-	80000
KL coef. start value	0	0	-	-	0
KL coef. end value	0	0.001	-	-	0.0001
KL coef. iterations	-	70000	-	-	70000
Commitment cost β*	-	-	0.25	0.25	-
EMA decay rate γ*	-	-	0.99	0.99	-
Num. downsamples	2	2	2	2	2
N_{RD}*	6	6	6	6	6
N_{RB}*	4	4	4	4	4
Hidden dim.	512	512	512	512	512

*τ - temperature parameter for the Gumble-Softmax in dVAE.
*β - Coefficient of the third term of VQ-VAE loss (in Eq. 1)
*γ - smoothing parameter used in EMA (in Eq. 6)
*N_{RD} and N_{RB} - num. residual blocks for downsample and bottleneck

Table 5: Training parameters for the discrete autoencoders.

of iterations. Thirdly, multi-step learning rate is used, so it is multiplied by LR γ on the epochs, that stated in "LR milestones" row.

The vocabulary utilization throughout training is shown on Figure 16. It is easy to notice, that without KL-divergence term in the loss function, dVAE model starts to degrade in terms of vocabulary utilization. Also, VQ-VAE model learns to use full vocabulary much faster with separate learning rate for embeddings. It was noticed, that it is especially important in the beginning of training.

Also, Figure 17 provides a qualitative example of image reconstruction for different models. The first column shows the original image and other five columns show the reconstruction made by VQ-VAE, VQ-VAE with separate learning rate for vocabulary, dVAE with linearly growing KL-divergence term in the loss, dVAE without KL-divergence term and dVAE trained on 64×64 images. It was noticed, that models learn to reconstruct the spatial structure of the image during the first two or three epochs, however, the colors can be a particularly difficult task. We will briefly go through the presented examples and discuss them.

The first and second rows clearly indicate, that dVAE had problems with the

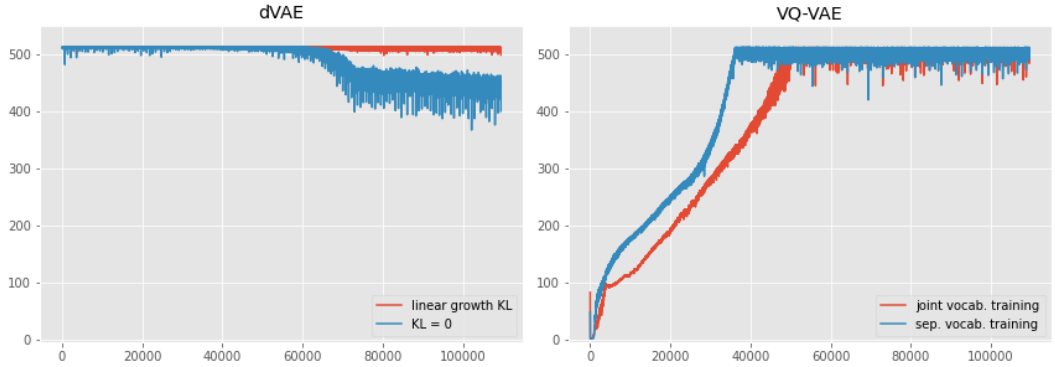


Figure 16: The usage of vocabulary during training for different types of discrete autoencoders. The CUB 128 × 128 dataset is used.

color reconstruction of the bird. We had an assumption that the problem might be connected to the presence of a particular color in the dataset, since red color is not very widespread in CUB, based o our observations. However, we can see, that the third row with red background on the original image was not properly reconstructed by any of the models (except dVAE, that was trained on lower resolution for a longer period of time). Also, there is a clear difference between the birds color for dVAE trained with and without KL-divergence term. It was also noticed during the experiments, that linear growth of this term helps better, than fixed value. In case of the last three rows, we can see that dVAE showed better results in terms of background color reconstruction. Especially it is easy to notice for the last row with black bird on the purple background, which turned entirely grey after the VQ-VAE reconstruction.

In general, it is not possible to make a clear conclusion about the better reconstruction ability of one model over another. Throughput the experiments we conducted, it seems that VQ-VAE model training requires more tricks and hacks in order to achieve high quality results in terms of image color. This observation is logically supported by the models architecture, since the VQ-VAE training goal is particularly more difficult in terms of the dimension of the latent space. It needs to learn to put the output vector of the encoder in the small area near the discrete embedding in the latent space, that is not restricted in any way. On the other hand, dVAE operates in the normalized space, where the choice is made by the higher value in the output logits, instead of all of them.

So, considering the logic above and the fact, that dVAE was originally used by the authors of the DALL-E model, we also exploited it as an image encoder for the generation experiments.

5.3 Image Generation Experiments

The experiments were conducted on two datasets - MD-MNIST and CUB. This section provides the parameters description of the models that were trained on these datasets and the examples of the generated images.

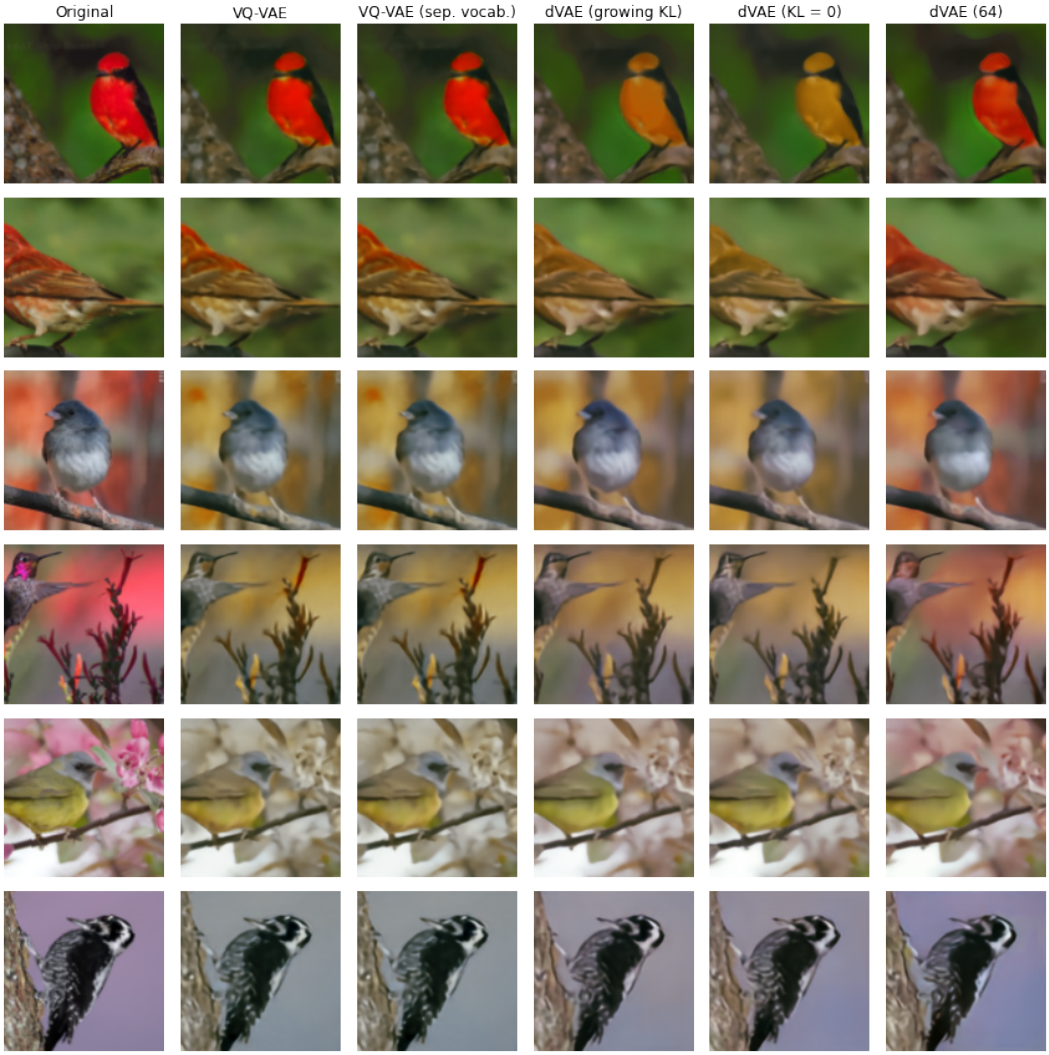


Figure 17: Reconstruction of samples from CUB dataset with different discrete autoencoders.

Section 5.3.1 is dedicated to comparison between two types of models, described above - with joint and separate text encoder. In the section 5.3.2 we discuss the models with separate text encoder trained on CUB dataset with different resolution. Such preference towards a particular architecture for this dataset will be explained in the next section.

5.3.1 MD-MNIST Generation

Two models were trained on MD-MNIST dataset - with separate text encoder and without one. We will further call these two models *G1* and *G2* respectively.

The training parameters for both models are presented in Table 6. We used Adam optimizer from the PyTorch package.

The images, generated by both models are presented on Figure 18 with the corresponding generation instructions. It is difficult to visually estimate which model

	$G1$ (joint encoder)	$G2$ (sep. encoder)
Transformer parameters		
Num. epochs	20	20
Batch size	12	12
Learning rate (LR)	0.001	0.001
LR γ	0.1	0.1
LR milestones	5,10	5,10
Num. encoder blocks*	12	12
Num. text encoder blocks	-	6
Num attention heads	8	8
DropOut prob.	0.1	0.1
Hidden dim.	512	512
dVAE parameters		
Num. epochs	30	30
Batch size	32	32
Learning rate (LR)	0.01	0.01
LR γ	0.1	0.1
LR milestones	5,10	5,10
τ* start value	5	5
τ* end value	0.01	0.01
τ* iterations	3000	3000
KL coef. value	0.01	0.01
Num. downsamples	2	2
N_{RD}*	4	4
N_{RB}*	4	4
Hidden dim.	256	256

*Num. encoder blocks - both text and image in case of *G1* .
*τ - temperature parameter for the Gumble-Softmax in dVAE.
*N_{RD} and N_{RB} - num. residual blocks for downsample and bottleneck

Table 6: Training parameters for the generative Transformer models and corresponding dVAEs for MD-MNIST dataset.

generates better pictures. So, the comparison analysis is presented in the section 5.4.

5.3.2 CUB Generation

For CUB dataset we used Transformers with separate text encoders. Initially, we tried to train the model on 128×128 dataset. However, since the model requires a lot of training time, we didn't reach the point with high generation quality. That's why we conducted the same experiment with the images of lower resolution 64×64.

The training parameters for both described models are presented on Table 7.

The training plots are presented on Figure 19. Since the loss curve has an

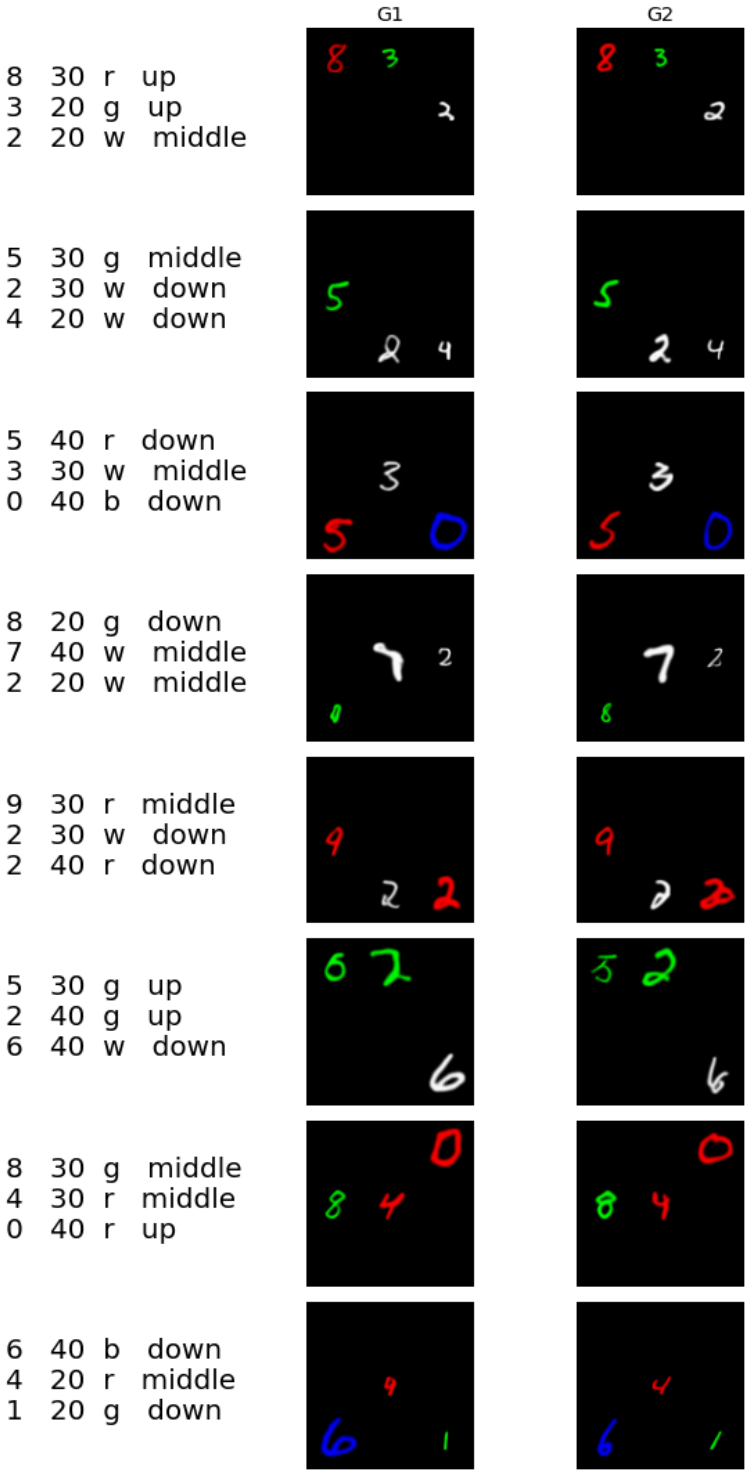


Figure 18: MD-MNIST generation with two types of Transformers. $G1$ (first column) stands for the model with joint text and image encoder and $G2$ (second column) for the separate one.

	CUB 64	CUB 128
Transformer parameters		
Num. epochs	1000	400
Batch size	120	32
Learning rate (LR)	0.001	0.001
LR γ	0.1	0.1
LR milestones	100,500	20
Num. image encoder blocks	14	12
Num. text encoder blocks	8	16
Num attention heads	8	8
DropOut prob.	0.1	0.1
Hidden dim.	512	512
dVAE parameters		
Num. epochs	500	250
Batch size	64	16
Learning rate (LR)	0.001	0.001
LR γ	0.1	0.1
LR milestones	20,120	5,100
τ* start value	8	5
τ* end value	0.01	0.01
τ* iterations	80000	140000
KL coef. start value	0	0
KL coef. end value	0.001	0.001
KL coef. iterations	70000	100000
Num. downsamples	2	2
N_{RD}*	6	6
N_{RB}*	4	4
Hidden dim.	512	512

*τ - temperature parameter for the Gumble-Softmax in dVAE.
*N_{RD} and N_{RB} - num. residual blocks for downsample and bottleneck

Table 7: Training parameters for the generative Transformer models and corresponding dVAEs for CUB dataset.

exponential behaviour, the begging of training is hidden to see the details of the later iterations. The transparent lines are the loss curves and the solid lines are MA[5] smoothed versions with window size of 100 iterations. The lines have different length due to the different number of epochs and batch size. As expected, the model, trained for 128×128 images has more volatile curve, but on average the 64×64 model reached lower values of loss in the end of training.

Figures 20, 21 and 22 provide the qualitative comparison of the generative abilities of the trained models.

[5]Moving Average

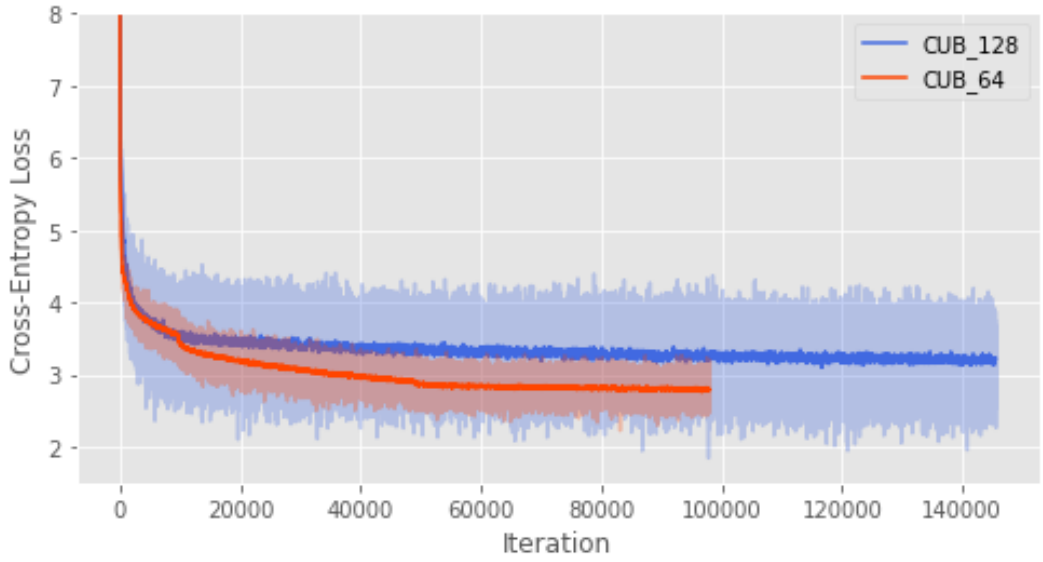


Figure 19: Loss curves for the described Transformer models, trained on CUB dataset with images of sizes 64×64 and 128×128.

The sample of six images, generated from caption "*the bird has a brown bill and yellow breast*", is presented on Figure 20. It is easy to notice, that the color of birds breast is yellow or close to yellow. Even on the images, where details are not very accurate, we can still observe yellow textures in the location corresponding to birds breast.

Figure 20: 64×64 images, generated from caption "*the bird has a brown bill and yellow breast*".

We can also examine a sample of birds images generated from different captions on Figure 21. The corresponding descriptions of the images are listed below. At some images the shapes were not generated properly, however, we can still see, that description correctly represents the content of the generated image. For example, it is clear, that birds on fifth and sixth images have white and yellow bellies respectively.

The source captions of the generated images presented on Figure 21:

- *this bird has wings that are black and had a long black bill*
- *this brown black bird has a very large wing span and a small head with a long pointed tail*
- *a small bird with black crown and blueish gray plumage*

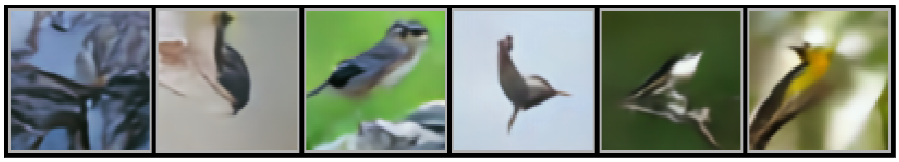


Figure 21: Generated 64 × 64 images. The captions are provided separately in the text of the current work.

- *this bird has wings that are black and has a long neck*
- *this bird has a white belly with a black back and head*
- *this is a very small bird with a yellow belly and a dark colored head and beak*

Additionally, we provide an example of the generated images from the model trained on 128 × 128 resolution.

As we can see, the model was able to catch the scene's textures. It is possible to recognise some parts of the bird, like legs or breast, in central part of the images. Second and sixth images even resembles the bird shape. Also, we can see, that even if the shapes are entirely broken, there is still some correspondence between the text and the generated image. For example, the forth image has a yellow spot in the central part and we can see, that the description mentions "*a yellow head*".

Figure 22: Generated 128 × 128 images. The captions are provided separately in the text of the current work.

The source captions of the generated images presented on Figure 22:

- *this bird has a very long reddish brown throat a white crown and a very long bill with brown and white feathers covering the rest of its body*
- *a small bird with a brown wing and tail and pointed beak*
- *small bird with a beautiful yellow throat breast and belly it has a black band that goes across its throat and brown cream speckled sides its tarsus are pink and it has a small beak*
- *a small bird with a yellow head face neck throat belly white tarsus grey rump and grey wings and tail with white wingbars and brown tips*
- *this particular bird has a black belly and gray breasts and a red bill with a black tip*
- *this bird has wings that are black and has a white belly*

5.4 Generative Models Comparison

The comparison was performed on MD-MNIST dataset. There are two reasons for that: firstly, as we showed above, the quality on CUB dataset is not high enough due to the limited resources. Secondly, MD-MNIST dataset is convenient for CTRS[6] metric calculation due to its artificial nature.

The first comparison was accomplished using cross-entropy loss comparison for the next token generation task given the previous correct tokens. The procedure and its drawbacks are described in the previous sections. The distribution of losses for both models described in the previous section is presented on Figure 23. Since the absolute majority of values was concentrated around zero, we used the log-scale for y-axis.

The average values of losses on 10000 samples are $loss(G1) = 0.3100$ and $loss(G2) = 0.3006$. So, we can not talk about significant difference in this case. The red tail of the distribution is a bit heavier than the blue one, which correspond to $G1$ and $G2$, respectively. However, it is still difficult to conclude superiority of one model over another.

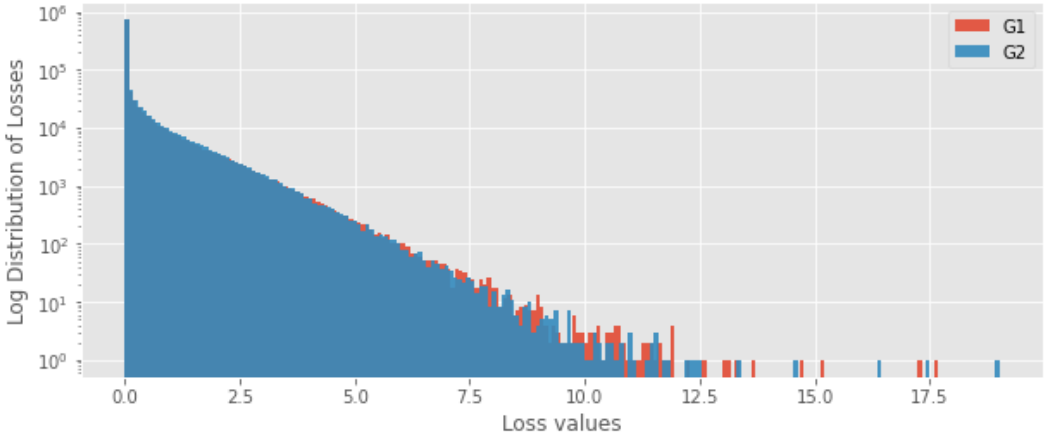


Figure 23: Log distribution of cross-entropy losses for next token prediction task.

CICS and CTRS metrics showed more representative comparison.

The first stage of such metrics calculation is CLIP training on the MD-MNIST dataset. It is known that image reconstructions using the models with discrete latents, like dVAE and VQ-VAE, is a little bit blured. So, in order to have unbiased training procedure for the CLIP, we trained the model on dVAE reconstructions instead of original images. The model is the same as the one described in Table 6.

For text and image representation we used Transformer architecture similar to ViT. Additionally to image patches or text embeddings, we added one more trainable embedding, that is considered to be the representation of the given sequence. We used 2D positional encoding for image encoder and 1D - for text encoder. Also, image patches were converted to a defined representation dimension with linear

[6]CLIP Text Ranking Score, described in section 4.3.3

transformation. The encoders outputs were linearly transformed to multi-modal dimension similarly to the original paper [10].

	CLIP Encoder parameters (both text and image)
Training	
Num. epochs	30
Batch size	64
Learning rate (LR)	0.001
LR γ	0.1
LR milestones	20
Architecture	
Num. encoder blocks*	8
Num attention heads	8
DropOut prob.	0.1
Representation dim.*	128
Hidden dim.	256

*Num. encoder blocks - number of Transformer encoder blocks.
*Representation dim. - the dimension of representation embedding.

Table 8: Training parameters for CLIP text and image encoders for MD-MNIST dataset.

The mentioned metrics were calculated using the trained CLIP.

We ran 546 iterations of comparisons between two architectures - $G1$ and $G2$. Each iteration every model generated 100 images. Then CICS was calculated for these 100 pairs and saved. So, we have 546 observations, where each observation is two numbers representing how many times CLIP chose one image over another in the given batch.

The quantitative results of the described comparison are presented in Table 9. There is a clear difference between overall CICS scores of two models - $G2$ was chosen more times than $G1$. The mean values across the batches also indicate better performance of $G2$ model.

In order to confirm the significant difference between two means, we conducted the statistical test for means between two samples of collected CICS metrics. The hypothesis of equal means is rejected on 10%-significance level. Also, the hypothesis that CICS mean is higher for $G1$ than for $G2$ is also rejected on 5%-significance level. The alternative can't be rejected on any reasonable significance level.

The log CTRS values distribution are presented on Figure 24. X-axis indicates the number of replacements performed on the description of the image. 0 is hidden from this figure, since it basically indicates the correct number of matches. As we can see, red bars are slightly higher than the blue ones. All the values are presented in Table 10.

	$G1$	$G2$
CICS	13546	13754
Mean CICS across batches	49.62	50.38
p-value (two-sided)	0.085	
p-value ($\overline{CICS(G1)} > \overline{CICS(G2)}$)*	0.042	
p-value ($\overline{CICS(G1)} < \overline{CICS(G2)}$)*	0.957	

*$\overline{x}$ - the average value of x.

Table 9: Quantitative analysis of CICS metric calculated on MD-MNIST dataset.

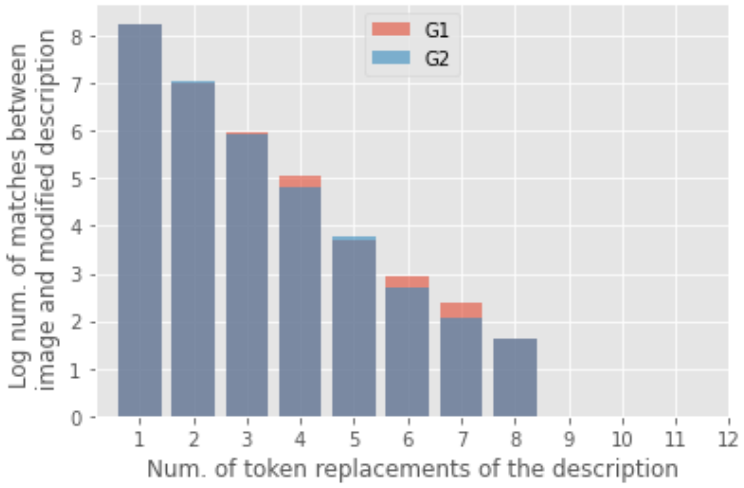


Figure 24: Log CTRS values.

	0	1	2	3	4	5	6	7	8	9	10	11	12
G1	21692	3858	1128	388	158	41	19	11	5	0	0	0	0
G2	21750	3833	1146	374	125	43	15	8	5	1	0	0	0

Table 10: Quantitative analysis of CICS metric calculated on MD-MNIST dataset.

6 Conclusions

In this work, we considered the task of conditional image generation given a textual description. The main focus is made on the autoregressive Transformer models for the text-to-image generation based on the discrete latent codes. We analysed different discrete autoencoder models and their performance in terms of vocabulary usage and its influence on image reconstruction quality. Also, we compared two DALL-E inspired architectures of the Transformer model - with and without separate text encoder. The conducted experiments suggest that there is a slight statistically significant advantage of the model with the separate text encoder.

In order to perform such comparison, we introduced two metrics, that are based on Contrastive Language–Image Pre-training. These metrics try to estimate the relevance of the produced images to the text from which they were generated. The comparison was performed on the artificially created dataset MD-MNIST, that was designed for convenient usage with the described metrics. However, this evaluation approach can be extended to measure the generation quality on the datasets with actual textual descriptions.

Finally, we trained the explored model with separate text encoder on the CUB dataset. Since we did not use any tricks, like mixed-precision training, and used only one GPU, we did not achieve high quality. However, the model learned to generate consistent images of birds, that are relevant to the given description.

Our work can be extended in several ways. The comparison between Transformers with and without separate text encoder can be performed on the real dataset, like CUB. As we mention above, the same metrics can be used for it. Also, the influence of different attention patterns can be explored for the architecture with separate text encoder. Additionally, we see a potential extension of the considered problem to the task of interactive image modification using the text instructions. To our knowledge, the works in this direction ([24], [8], [28]) use the continuous hidden representation of the images. The usage of discrete representation, considered in this work, can significantly reduce the volume of the latent space and allow to formulate the problem as a question “what image tokens should we replace to make the image relevant to a given text?”. Such approach can potentially provide a high interpretability and control over the generation process. The described questions and ideas are left for further research.

References

[1] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In Advances in Neural Information Processing Systems, pages 4790–4798, 2016

[2] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

[3] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. CoRR abs/1609.03499, 2016.

[4] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, Ilya Sutskever. Zero-Shot Text-to-Image Generation. *arXiv:2102.12092*, 2021.

[5] Adrian Lancucki, Jan Chorowski, Guillaume Sanchez, Ricard Marxer, Nanxin Chen, Hans J.G.A. Dolfing, Sameer Khurana, Tanel Alumäe, Antoine Laurent. Robust Training of Vector Quantized Bottleneck Models. In *IJCNN 2020*, 2020.

[6] Ali Razavi, Aaron van den Oord, Oriol Vinyals, Generating Diverse High-Fidelity Images with VQ-VAE-2. *arXiv preprint arXiv:1906.00446*, 2019.

[7] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In Advances in Neural Information Processing Systems, Annual Conference on Neural Information Processing Systems, NeurIPS, 2017.

[8] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, Philip H. S. Torr. ManiGAN: Text-Guided Image Manipulation. *arXiv preprint arXiv:1912.06203v2*, 2019.

[9] Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

[10] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[11] Dieleman, S., van den Oord, A., and Simonyan, K. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pp. 8000–8010, 2018.

[12] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. CoRR, abs/1409.0473, 2014.

[13] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, Dimitris Metaxas. StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks. *arXiv preprint arXiv:1710.10916*, 2018.

[14] He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016a.

[15] Jang, E., Gu, S., and Poole, B. Categorical Reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122v2*, 2017.

[18] Kingma, D. P., & Welling, M. Auto-encoding variational bayes. *ArXiv Preprint ArXiv:1312.6114.*, 2013

[19] Oord, A. v. d., Vinyals, O., and Kavukcuoglu, K. Neural discrete representation learning. *arXiv preprint arXiv:1711.00937*, 2017.

[20] Patrick Esser, Robin Rombach, Björn Ommer, Taming Transformers for High-Resolution Image Synthesis. *arXiv preprint arXiv:2012.09841*, 2021.

[21] Prafulla Dhariwal, Heewoo Jun, Christine Payne, Jong Wook Kim, Alec Radford, Ilya Sutskever, Jukebox: A Generative Model for Music. *arXiv preprint arXiv:2005.00341*, 2020.

[22] Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G., and Sutskever, I. Learning transferable visual models from natural language supervision. 2021.

[23] Ruslan Rakhimov, Denis Volkhonskiy, Alexey Artemov, Denis Zorin, Evgeny Burnaev, Latent Video Transformer. *arXiv preprint arXiv:2006.10704* , 2020.

[24] Seonghyeon Nam, Yunji Kim, Seon Joo Kim. Text-Adaptive Generative Adversarial Networks: Manipulating Images with Natural Language. *arXiv preprint arXiv:1810.11919*, 2018.

[25] Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

[26] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, Xiaodong He. AttnGAN: Fine-Grained Text to Image Generation with Attentional Generative Adversarial Networks. CVPR. 2018.

[27] Yifan Jiang, Shiyu Chang, Zhangyang Wang. TransGAN: Two Pure Transformers Can Make One Strong GAN, and That Can Scale Up. *arXiv preprint arXiv:2102.07074v1*, 2021.

[28] Yu Cheng, Zhe Gan, Yitong Li, Jingjing Liu, Jianfeng Gao. Sequential Attention GAN for Interactive Image Editing. *arXiv preprint arXiv:1812.08352v4*, 2020.